Analysis of kNN Model Performance on the MNIST Dataset

Alex Angus and Zach Nussbaum

October 2019

Introduction

Imagine that you are a data scientist for the US postal service and you want to create an address detector so that mail can be sorted using machine learning techniques. To accomplish this, we set out to create a digit classifier, using the MNIST dataset, which a dataset consisting of handwritten images (1). To construct this, we utilized a K-Nearest Neighbor classifier to denote the different digits from $0 - 9$.

Theory

The K-Nearest Neighbor (KNN) is a non-parametric method that can be used for classification and regression problems (2). This method utilizes learning where no actual learning is performed on the training set, and instead the data is stored. When predicting, the classifier predicts the output by calculating the distance the test input is from all the training data. The closest neighbors perform a voting where k is the number of neighbors to consider. While only looking at the k closest neighbors, the classifier votes on the output by taking the maximum number of outputs for a certain label. This is the standard KNN classifier. There are other variations of the classification, such as a weighted KNN, where the votes are calculated by summing $1/d$, where d is the distance from the current test example.

Experiments

In our analysis, we chose to implement 4 different kNN classification models. Our models are:

> **Unweighted Euclidean kNN:** A simple k-nearest-neighbors algorithm with a Euclidean distance metric and unweighted neighbor predictions. The Euclidean distance d_E between a test example x_0 and a training example x_j is defined as

$$d_E = \sqrt{\sum_{i=1}^{n} (x_{ji} - x_{0i})^2}$$

where x_{ji} is a component of the feature vector $\vec{x_j} = (x_{j1}, x_{j2}, \ldots, x_{jn})$ and n is the number of features. In the unweighted neighbor model, the k nearest neighbors are given an equal number of 'votes' in determining the prediction value.

Weighted Euclidean kNN: The Euclidean kNN model, with closer neighbors having a larger influence (weight) on the prediction. In our case we define the weight w_k for a given k^{th} neighbor to be

$$w_k = \frac{1}{d_k}$$

where d_k is the distance from our test example to the k^{th} nearest neighbor. In the weighted neighbor model, the influence of the k nearest neighbor votes are scaled by this weight.

Unweighted Manhattan kNN: The unweighted kNN model, with a Manhattan distance metric in place of the Euclidean distance. The Manhattan distance d_M is defined as

$$d_M = \sum_{i=1}^{n} |x_{ji} - x_{0i}|$$

For this model we implement the unweighted prediction method.

Weighted Manhattan kNN: In this model we use the weighted prediction model and the Manhattan distance metric together.

Data Trimming and Resolution Based Feature Selection

Our dataset is comprised of tens of thousands of hand written, 28 by 28 pixel images. This corresponds to 784 pixels per image, or 784 features per training example. The kNN algorithm is already very computationally expensive, and therefore it is not reasonable to design our model based on the full length and resolution dataset. We therefore implement the following methods to reduce the complexity of our training set:

Data Trimming: This method is simple. We simply take only a subsection of our data to train[1] with. For the sake of time, we choose to trim our dataset with length on the order of 10^4 by a factor of about 10, leaving us with about 10^3 training examples. Note that the data is shuffled beforehand to avoid any positioning bias.

[1] We are not doing any real 'training' with the kNN algorithm. By training we mean the process of predicting from the training set.

Figure 1: Comparison between 28x28 pixel digit (left) and 15x15 pixel digit (right).

Figure 2: Comparison between 28x28 pixel digit (left) and 8x8 pixel digit (right).

Resolution Based Feature Selection: Another method of reducing the size of our dataset is feature selection. In our case, one feature corresponds to a single pixel position. Therefore, to reduce the number of features, we must reduce the number of pixels. In order to reduce the number of pixels without losing all of the information of the pixels we eliminate, we implement a method[2] that takes the average value of a group of pixels, and replaces those pixels with a larger, averaged pixel value. Examples of our resolution reduction can be seen in Figure 1 and Figure 2. Through trial and error, we found our models to perform optimally at a resolution of 8x8 pixels.

[2]Specifically, we use the Image.resize function of the PIL module.

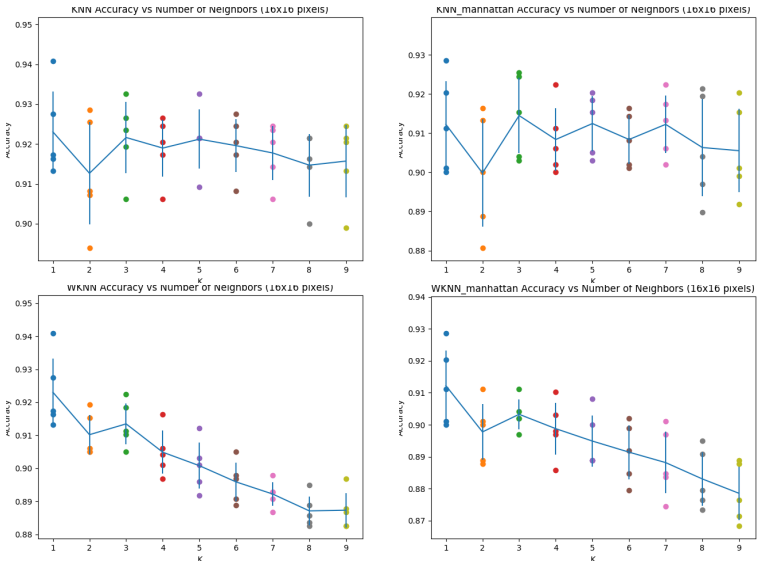


Figure 3: Accuracy and Error for each of our different models for the 16x16 images.

Results

To evaluate the performance of our models on the MNIST dataset (reduced to size 16x16), we implement k-fold validation with $k = 5$, and a hyperparameter sweep considering from 1 to 9 nearest neighbors. The number of neighbors K with the highest mean accuracy in the k-fold validation trial is the value we take as K in the final testing of each model, respectively. Plots depicting the mean accuracy of these trials can be seen in Figure 3.

After choosing a K value for each of our models, we then look at the confusion matrix of each model's performance on the test set. The confusion matrices of each model can be seen in Figure 4. An accurate model will have a confusion matrix of ones on diagonal, and zeros at almost all other spots in the matrix. This is what we observe with out models, therefore we can deduce that they are functioning.

We choose also to examine the specific mistakes that our model is making. To do this, we examine the error matrix, which is essentially a confusion matrix without the main diagonal indication. The error matrices for our 4 models can be seen in Figure 5-8. It is clear from the error matrices that our models are performing almost identically. A bright yellow spot on the error matrix indicates a consistently miscategorized digit. We see the same miscategorization of 9s as 4s, and vice-versa in each model. To gain a deeper understanding of why this is happening, we need to look at the digits themselves. In Figure 9, we have the miscategorizations of 4s and 9s made by the KNN model. On top of 4s and 9s having similar figures, if we examine the digits that were miscategorized, it is

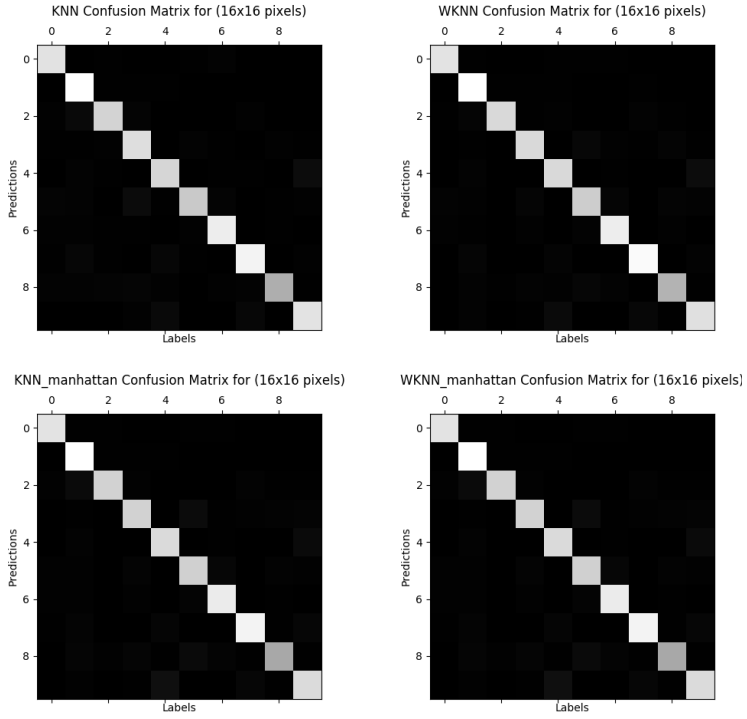


Figure 4: Confusion Matrices of our 4 models. From the confusion matrices we see that the four models are performing well.

Figure 5: Error Matrix for KNN, k=3

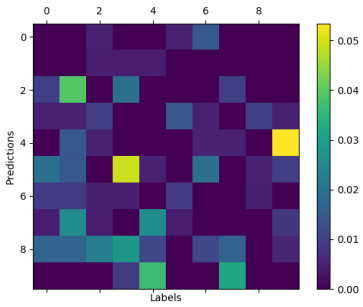
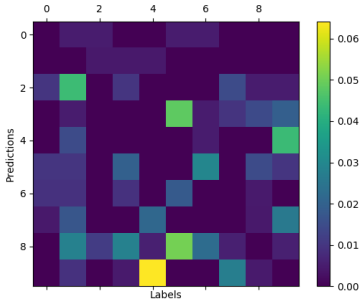


Figure 6: Error Matrix for KNN Manhattan, k=3



clear that they were poorly written digits to begin with. Even a human could have trouble classifying these digits, and thus if we are to further improve the performance of our models, we must first be given an improved set of training data.

We are, however, confident in the integrity of our models as we minimized overfitting through cross validation.

Choosing a Model:

Because our models are so closely performing in terms of accuracy, we choose our distance metric based on time performance. The Euclidean distance metric was significantly more time efficient in completing the kNN algorithm than the Manhattan distance, so we choose **Euclidean** as our distance metric. In terms of prediction weights, we observed no significant improvement in weighted prediction versus non-weighted prediction model performance. We therefore choose the simple, **unweighted** prediction process. Our final model is then the **Unweighted Euclidean kNN** with $k = 3$. Our test results using our best model resulted in an accuracy of 93%, which is slightly worse than other

Figure 7: Error Matrix for WKNN Manhattan, k=1

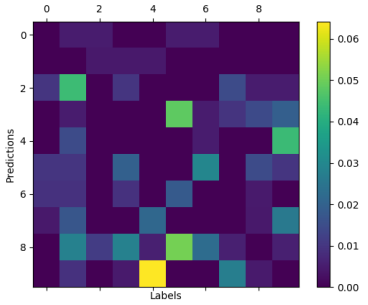
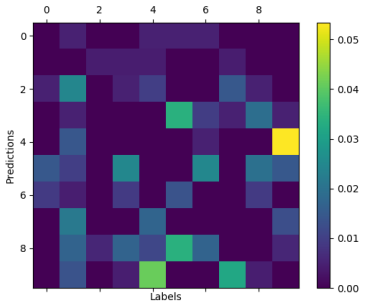


Figure 8: Error Matrix for WKNN, k=1



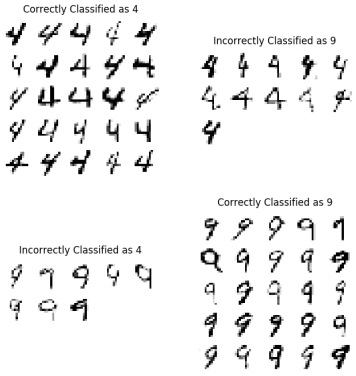


Figure 9: Miscategorized predictions for the KNN model

reported results.

Contributions

Alex: Experiment and Results sections, fetching MNIST data, Feature Reduction (Resolution) and trimming functions, Confusion Matrix plots, Error Matrix plots, plots of miscategorized digits, Manhattan Distance/Weighted Manhattan models.

Zach: Intro, Theory, Experiment and Results sections, normalization, KNN and weighted KNN, KFold Cross Validation, Accuracy and Error plots, Manhattan Distance/Weighted Manhattan models, train/test split/shuffling, testing function

References

[1] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[2] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46(3), 175-185.